

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

I recently started using a network tool called Ethereal. For those familiar with tcpdump, think of Ethereal as a GUI form of tcpdump that shows you the whole packet and can break down the packet to show individual fields. For those who haven't used tcpdump or similar packet sniffers, it might be best to show the capabilities of Ethereal through a few examples.

When you start Ethereal, it looks like the graphic shown in Figure 1. Typically, you want to capture some data from the network attached to your workstation; do this by selecting Capture→Start..., which brings up the dialog shown in Figure 2. When you've captured the data you need, stop the capture and examine it. Figure 3 shows a capture of some IPv6 traffic, where I've selected an ICMPv6 packet (in the top frame) and expanded the IPv6 and ICMPv6 contents to select the IPv6 source address (in the middle frame). Ethereal automatically highlights the raw bytes corresponding to the selected field—in this case, source address—within the packet in the bottom frame. This type of functionality makes Ethereal useful for understanding various network protocols, and I definitely recommend its use as a teaching or self-education aid in conjunction with networking RFCs. Ethereal also is useful for educating users and management about the dangers of using protocols that send data in clear text, as shown for File Transfer Protocol in Figure 4.

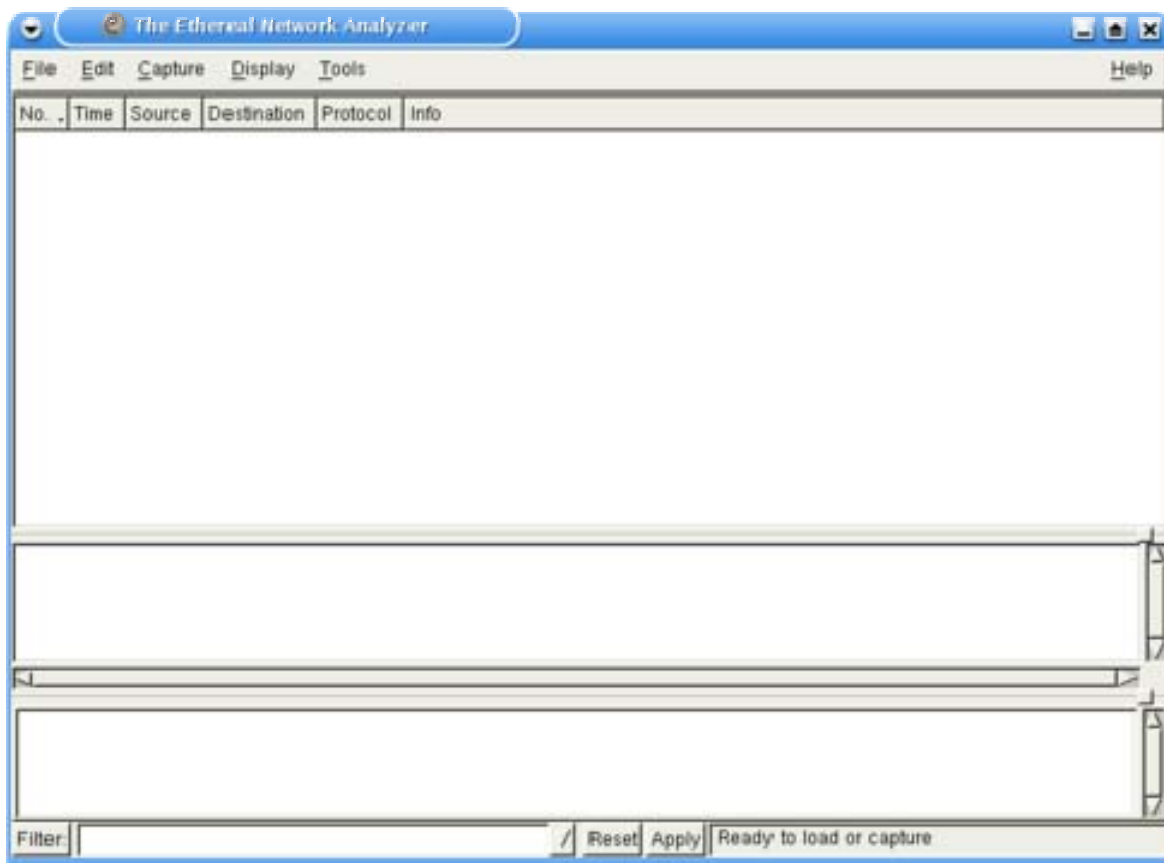


Figure 1
The Ethereal Main Window

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

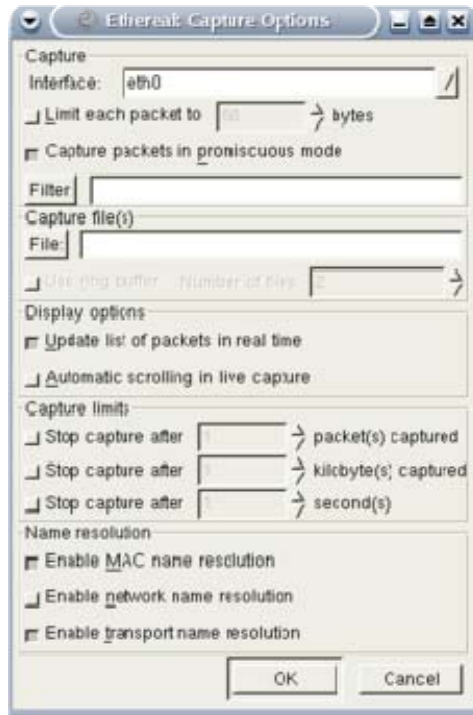


Figure 2
Capture Dialog

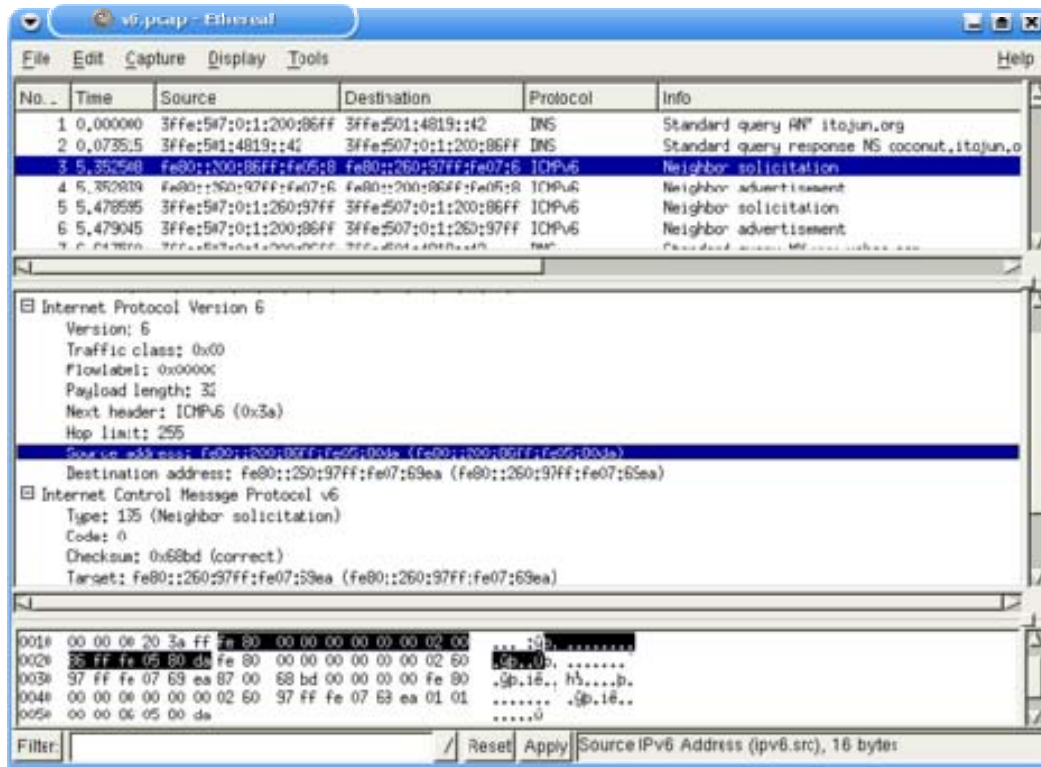


Figure 3
ICMPv6 Capture and Dissection

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

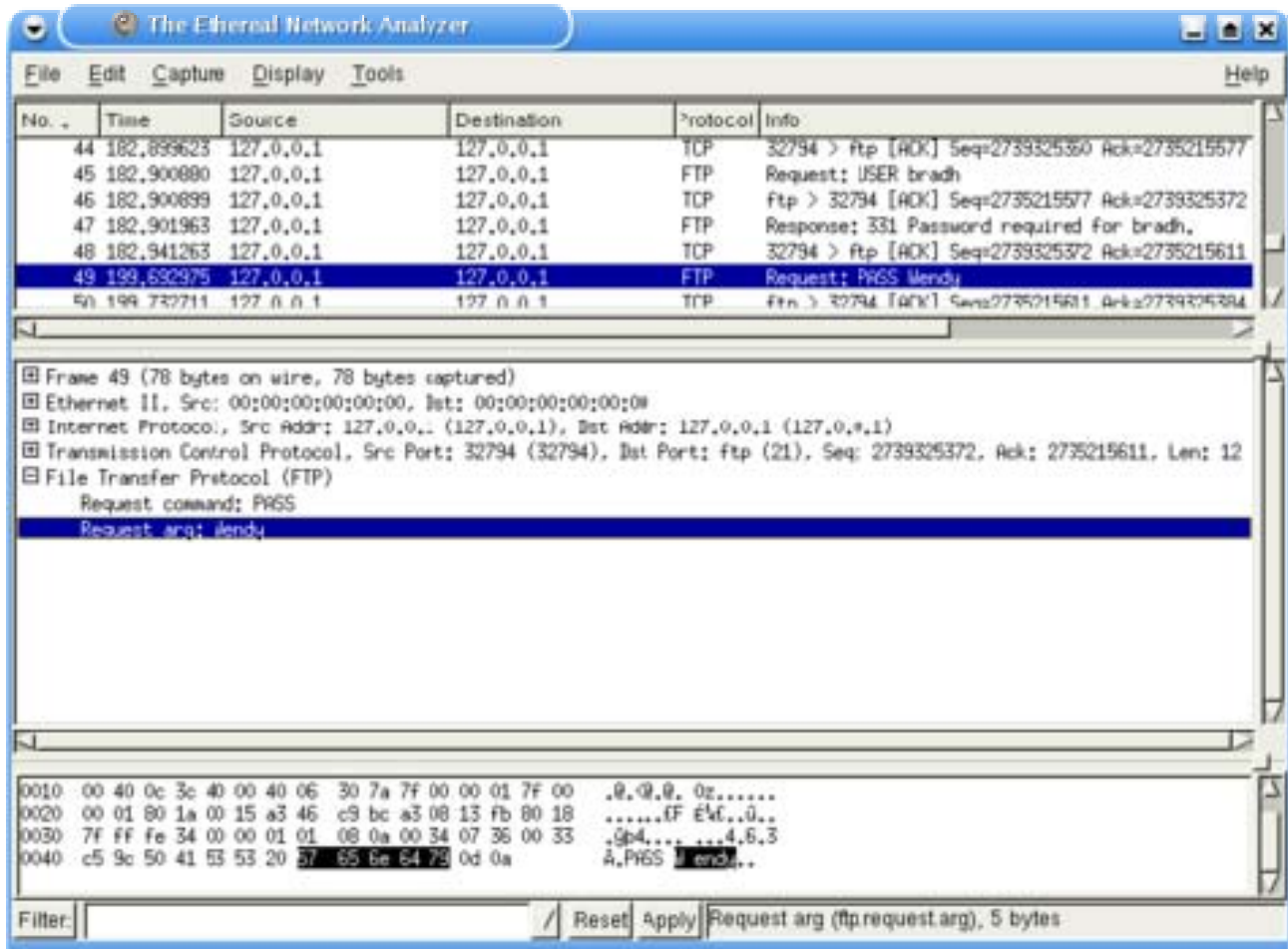


Figure 4
FTP Capture and Dissection, Showing Password

Ethereal also is useful for investigating proprietary protocols or other networking protocols that are not well documented. Figure 5 shows a somewhat contrived example—rsync. This protocol is in widespread use because of its ability to save significant bandwidth but is essentially defined by the source code to the application. I used Ethereal to capture a number of rsync transactions and figured out how the protocol works—at least enough to write an rsync protocol dissector for Ethereal. I understand the Samba team uses Ethereal and a number of other tools to develop clients and servers that interoperate with the Microsoft CIFS implementations, because the Microsoft documentation for these protocols is incomplete or incorrect.

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

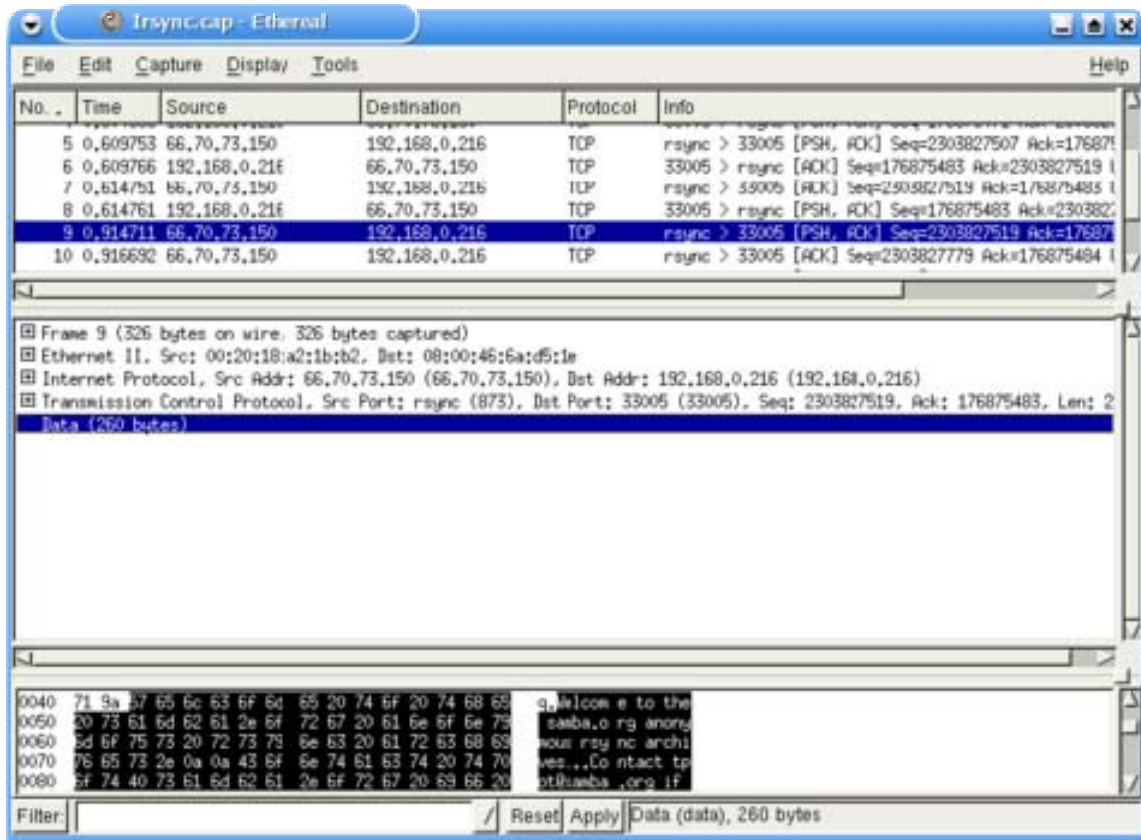


Figure 5
Ethereal Capturing rsync (Now Supported)

I also have used Ethereal as a part of network application testing (on zcip and Service Location Protocol) to assess correctness and response times. Ethereal time-tags each transaction, so you easily can see the relationship between packets.

How Ethereal Works

Ethereal works by capturing packets through a reasonably portable library called libpcap, which on Linux accesses the packets on the network through using a kernel mechanism called packet socket. It is possible to disable this option under Linux, although probably all vendor kernels have it enabled, and it is enabled in the default kernel configuration for most architectures on Linux kernels. Other operating systems have different interfaces, but libpcap abstracts this away and provides a common API.

Having received a copy of the network packets, Ethereal builds an internal linked list and saves the packets to a file. It then determines what protocol the packet is carrying based on the port numbers, type fields in the supporting protocols or a heuristic that guesses the protocol based on the contents of the field. It is worth noting that this approach essentially is informed guesswork and is by no means infallible. For example, traffic to port 53 probably is DNS, but there is no reason why a network administrator could not choose to run another service on that port. In addition, Ethereal supports an option to interpret a particular packet as a different protocol, using Tools→Decode As.

Based on the guessed protocol, Ethereal decodes (dissects, in Ethereal nomenclature) the packet. Each protocol supported by Ethereal is handled through a bit of code known as a dissector. At the

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

time of this writing, 333 dissectors are built in to Ethereal, some of which handle more than one protocol. Protocols also can be provided as plugins, which are loaded dynamically. Depending on the protocol and the level of sophistication provided by the dissector code, the packets can be broken down for analysis of individual bits or they can be presented at a very high level. Both options are depicted in Figure 6, where the TCP dissector shows the individual bits set in the flags, but the IMAP dissector breaks out only two fields. It is worth noting that IMAP is a text-based protocol, so a simple ASCII dump of the packet contents is an appropriate way to show them.

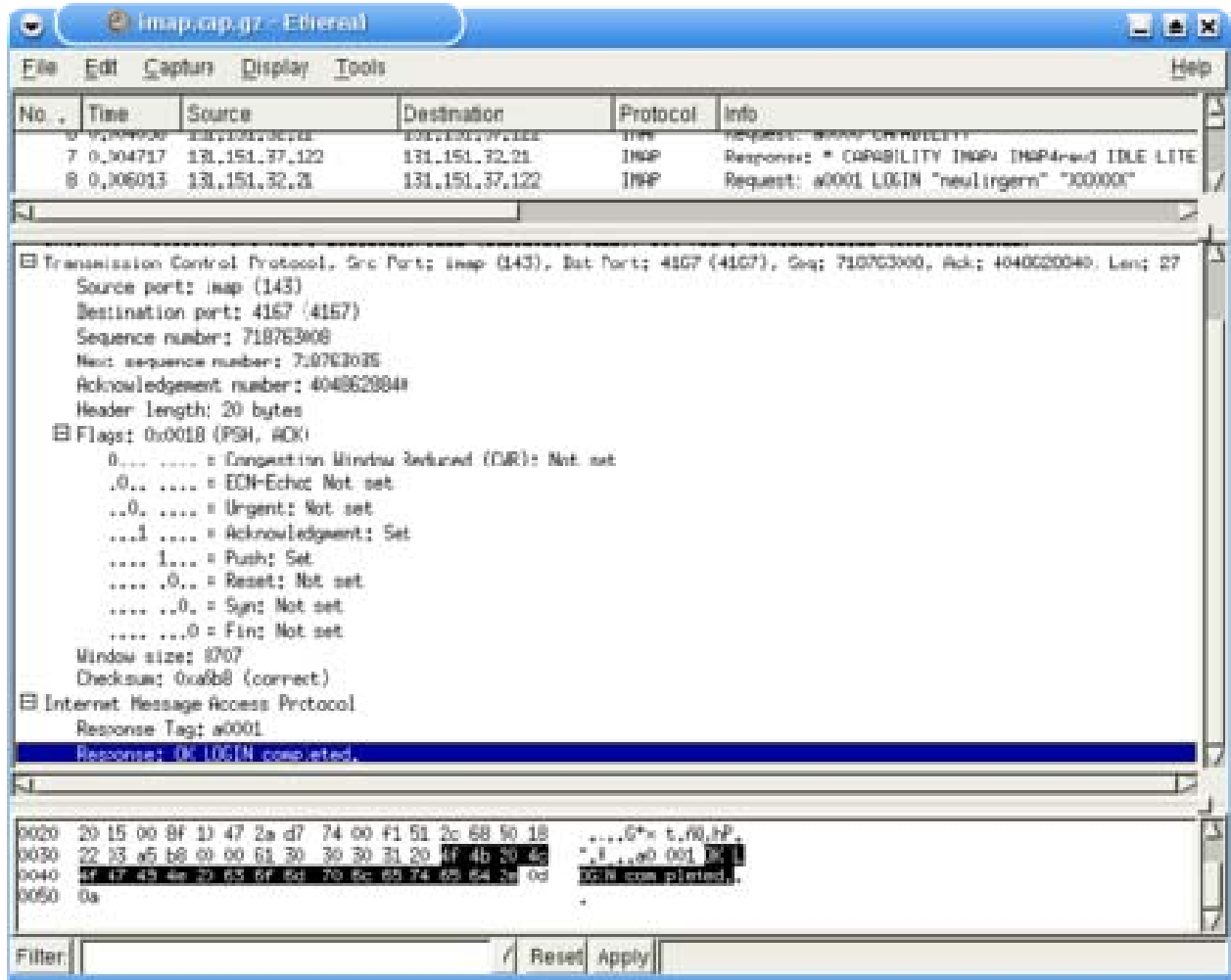


Figure 6
Two Variations on Dissection—TCP and IMAP

Key Features

From my point of view, the key features of Ethereal are its ability to capture and analyze network traffic within a single application and the sophistication of its display and filtering code.

Although we looked earlier in this article at how capturing network traffic is done, Ethereal can capture more than Ethernet traffic. Ethereal typically can (at least on Linux) capture data from Ethernet, Token-Ring, FDDI, serial (PPP and SLIP), 802.11 wireless LAN, ATM connections and all networking devices at the same time. Called the “any” device in the Ethereal capture dialog, this feature only

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

works in Linux. Of course, suitable networking hardware and kernel drivers need to be enabled to get the packets.

On a busy network, you may have thousands of packets in a capture file and be interested in only some of them. To make it easier to interpret the Ethereal display, which can get pretty busy, you can use colors. From the Display→Colorize Display... option, you can select display packets in various colors; Figure 7 shows how the filter is specified. In this case, I'm filtering on only a single field (the version number for Service Location Protocol), but you can build sophisticated filters with Boolean logic. Figure 8 shows a typical example with a few filters, and Figure 9 shows the working display (with Service Location Protocol Version 2 in red, DNS in green and ARP in blue). You can use a wide range of text colors as well as background coloring to separate out the various protocols.

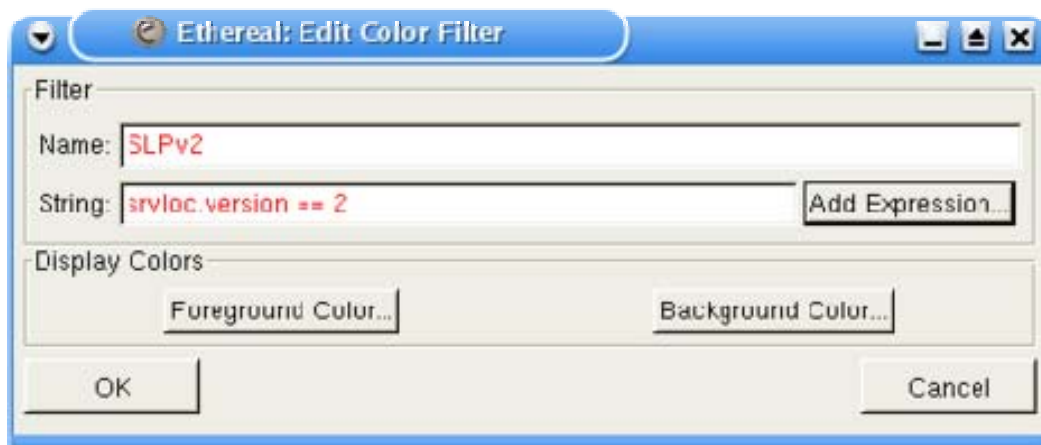


Figure 7
Specifying an Ethereal Color Filter

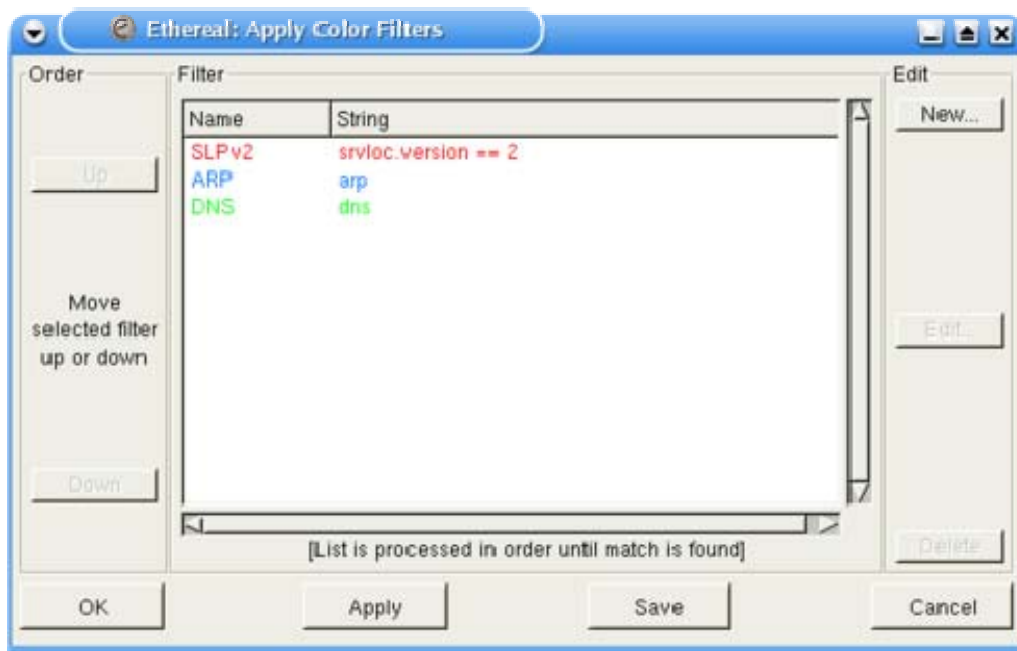


Figure 8
Ethereal Supports Multiple Color Filters

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

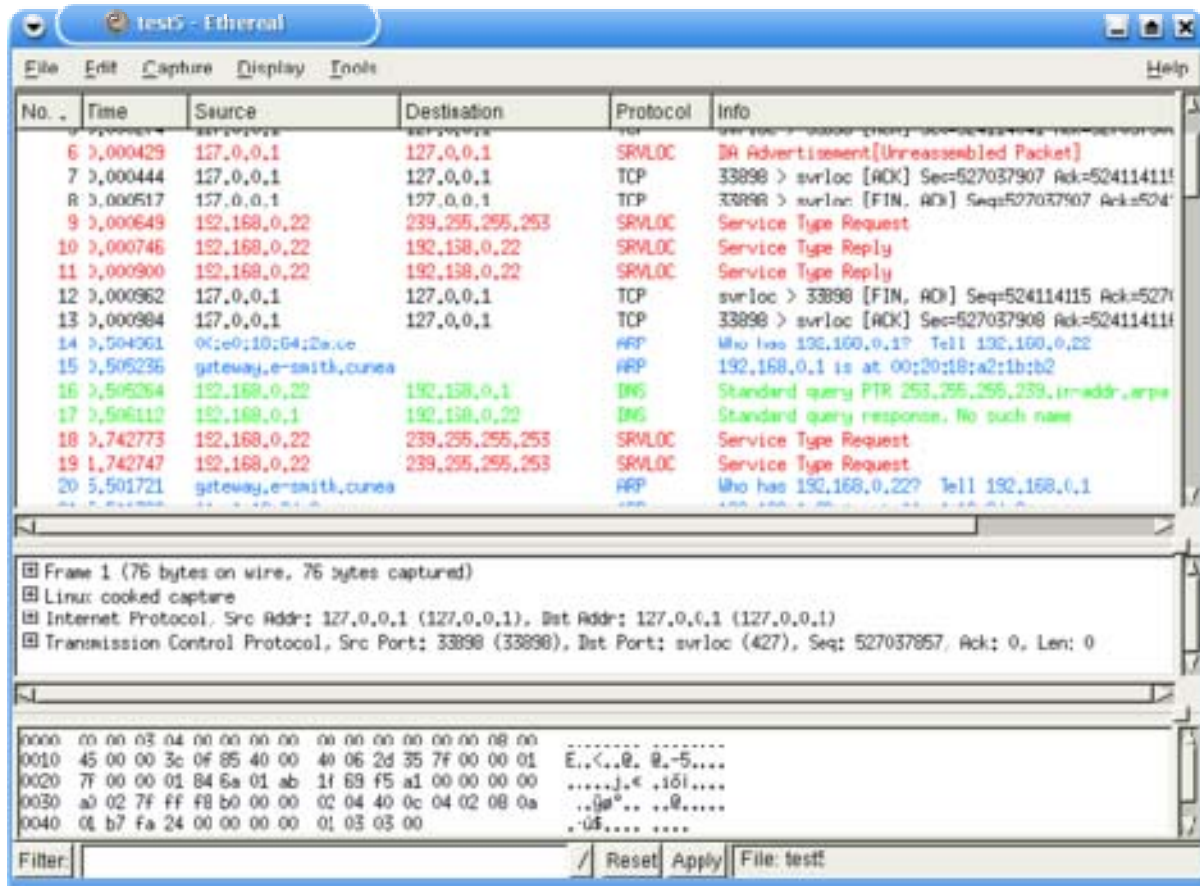


Figure 9
A Typical Colored Capture Session

After coloring the display, the next step is to remove packets of no interest, a task Ethereal handles through display filtering. A simple example is shown in Figure 10, where adding a svrloc filter (in the bottom left of the window) has removed all the other protocols, leaving only the Service Location Protocol. If this still is too complex, you could choose to change the coloring again, this time showing packets from particular hosts in separate colors or packets containing particular types of client requests or server responses in particular colors.

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

No.	Time	Source	Destination	Protocol	Info
4	0.000255	127.0.0.1	127.0.0.1	SRVLOC	Service Request
6	0.000429	127.0.0.1	127.0.0.1	SRVLOC	DA Advertisement[Unreassembled Packet]
9	0.000649	192.168.0.22	239.255.255.253	SRVLOC	Service Type Request
10	0.000746	192.168.0.22	192.168.0.22	SRVLOC	Service Type Reply
11	0.000900	192.168.0.22	192.168.0.22	SRVLOC	Service Type Reply
18	0.742773	192.168.0.22	239.255.255.253	SRVLOC	Service Type Request
19	1.742747	192.168.0.22	239.255.255.253	SRVLOC	Service Type Request
31	75.149237	192.168.0.22	239.255.255.253	SRVLOC	Service Request
40	90.142666	192.168.0.22	239.255.255.253	SRVLOC	Service Request
44	99.487553	127.0.0.1	127.0.0.1	SRVLOC	Service Registration
45	99.487960	127.0.0.1	127.0.0.1	SRVLOC	Service Acknowledge
48	99.488539	127.0.0.1	127.0.0.1	SRVLOC	Service Registration
49	99.488843	127.0.0.1	127.0.0.1	SRVLOC	Service Acknowledge
50	99.489326	127.0.0.1	127.0.0.1	SRVLOC	Service Registration
51	99.489620	127.0.0.1	127.0.0.1	SRVLOC	Service Acknowledge
52	99.489975	127.0.0.1	127.0.0.1	SRVLOC	Service Registration

Filter: srvloc / Reset Apply File: test5

Figure 10
Display Filtering on Same Session as Shown in Figure 9

Another option is to not capture the unwanted packets in the first place. To do this, Ethereal supports the same capture filter syntax that tcpdump uses. An example of this syntax is shown in Figure 11, where the dialog captures only the packets going to or from the machine with IP 192.168.0.1. Unfortunately, the syntax used in capture filters is different from that used in the display filters, a fact that makes capture filtering much less accessible to occasional users.

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

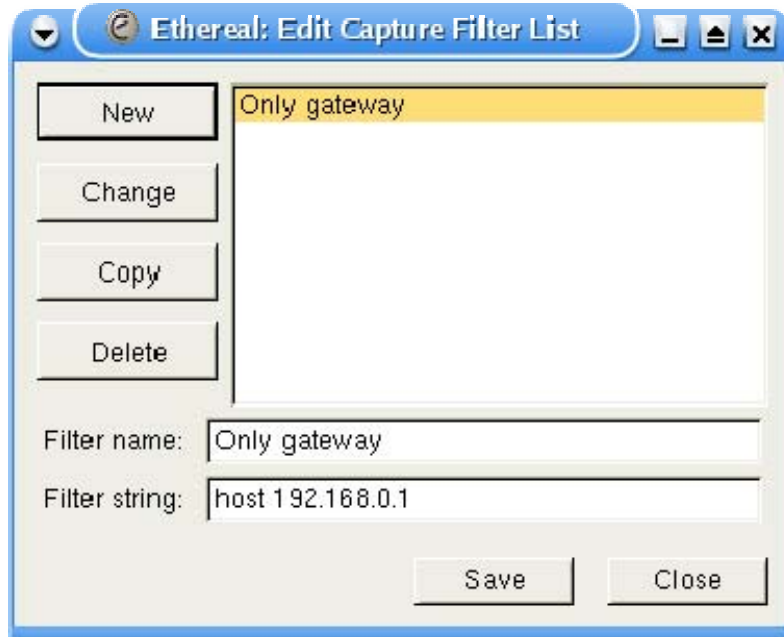


Figure 11
Capture Filtering Dialog

Another feature that some people find useful is the Follow TCP Stream... tool, which presents a text representation of the conversation. I personally don't use this feature often, but it is a powerful tool for looking at text-based protocols such as IMAP (Figure 12).

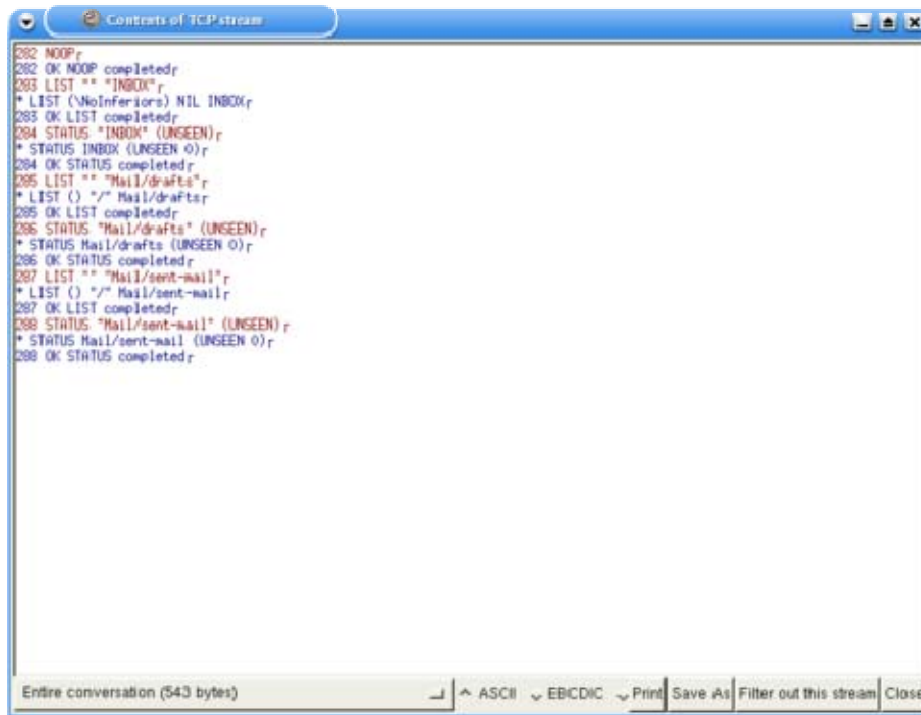


Figure 12
Following a TCP Stream—IMAP

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

Misfeatures and Omissions

Apart from the different syntaxes required for capturing and displaying filters, I've come across a few other issues in the time I've been using Ethereal. Some of these have to do with personal preferences, and others have been gleaned from monitoring the Ethereal mailing lists.

At the time of this writing, my biggest issue is with the quality of the support documentation, especially the User's Guide, which is incomplete and outdated. Also, a significant amount of the User's Guide, about the last 80%, is generated automatically and is not user-friendly. In addition, the version on the Web site has not been regenerated in some time. I personally found the GUI a little difficult to get used to, although as I became more familiar with the various menus, I became more productive with Ethereal. Perhaps some better documentation would have helped with this. There is also limited developer documentation, although I see this as a less important issue, given the large number of examples from which you can work.

Various users occasionally ask "when will such and such a protocol be supported?" Where I have found a few protocols not supported by Ethereal (rsync, distcc and ACAP), I've generally needed to code support myself. This is fairly easy to do with Ethereal. If you need support for a particular protocol, however, and it is not supported by Ethereal at the moment, you should allow for some development effort (either as an in-house development or on a contract basis) before committing to Ethereal. If you do develop additional dissectors or enhance an existing one, I strongly recommend that you have it incorporated into the Ethereal source tree to ensure it remains up to date.

Another feature supported by other packet analysis tools is the ability to capture data on a remote host and display it locally. If you can run Ethereal on the remote host, this scenario is possible, but often you want to capture data on a machine acting as a router or a server, where a full-blown X environment is undesirable. This lack may be overcome in a future version or it may not be particularly important, depending on your environment.

The only other issue worth mentioning is that a substantial number of the queries on the user-support mailing list seem to be from Windows users experiencing a wide range of problems. I personally haven't run the Windows version, so I don't know if the difficulties are associated with the underlying tools (especially WinPcap), Windows itself or the skill levels of the users.

Resources

For more information on Ethereal, start with www.ethereal.com. This page includes links to the Ethereal manual, downloads and mailing lists.

To understand better what Ethereal is showing you, you need the appropriate documentation on the network protocol. Those protocols, codified by the Internet Engineering Task Force, are available at www.rfc-editor.org.

rsync is an efficient network file transfer application originally developed by Andrew Tridgell (of Samba fame). See rsync.samba.org.

zcp is a tool for automatic assignment (zeroconf) of IPv4 addresses, without needing a DHCP server. See zeroconf.sourceforge.net.

Service Location Protocol is a way for clients to find servers in a network-efficient way. See www.srvloc.org for more details on the protocol, or refer to RFC 2608, RFC 2609, RFC 2610 and RFC

A Guided Tour of Ethereal

By Brad Hards
(Reprinted From The Linux Journal)

2614, available from www.rfc-editor.org. A free implementation, mainly developed by Matt Peterson, is available at www.openslp.org.

The capturing capabilities of Ethereal depend on libpcap, developed by the TCPDUMP Group. You need libpcap to build Ethereal, although most distributions ship with libpcap packages. See www.tcpdump.org.

The Windows version of libpcap is WinPcap. See winpcap.polito.it for more information and to download the installer package.

IMAP (Internet Message Access Protocol) is a server-based e-mail protocol, in many ways superior to the Post Office Protocol (POP) that is widely used. For more details, get RFC 2060 from www.rfc-editor.org.

distcc is a distributed compilation application developed by Martin Pool that uses various network machines to participate in building C code. See distcc.samba.org.